Linear Independence Oracles and Applications to Rectangular and Low Rank Linear Systems

Arne Storjohann
astorjoh@uwaterloo.ca

Shiyun Yang
s97yang@uwaterloo.ca

David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada N2L 3G1

ABSTRACT

Randomized algorithms are given for linear algebra problems on an input matrix $A \in \mathsf{K}^{n\times m}$ over a field K. We give an algorithm that simultaneously computes the row and column rank profiles of A in $2r^3 + (r^2+n+m+|A|)^{1+o(1)}$ field operations from K, where r is the rank of A and $|A|$ denotes the number of nonzero entries of A. Here, the $+o(1)$ in cost estimates captures some missing $\log n$ and $\log m$ factors. The rank profiles algorithm is randomized of the Monte Carlo type: the correct answer will be returned with probability at least $1/2$. Given a $b \in \mathsf{K}^{n\times 1}$, we give an algorithm that either computes a particular solution vector $x \in \mathsf{K}^{m\times 1}$ to the system $Ax = b$, or produces an inconsistency certificate vector $u \in \mathsf{K}^{1\times n}$ such that $uA = 0$ and $ub \neq 0$. The linear solver examines at most $r+1$ rows and r columns of A and has running time $2r^3 + (r^2+n+m+|R|+|C|)^{1+o(1)}$ field operations from K, where $|R|$ and $|C|$ are the number of nonzero entries in the rows and columns, respectively, that are examined. The solver is randomized of the Las Vegas type: an incorrect result is never returned but the algorithm may report FAIL with probability at most $1/2$. These cost estimates are achieved by making use of a novel randomized online data structure for the detection of linearly independent rows and columns.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*algebraic algorithms, analysis of algorithms*; G.4 [**Mathematical Software**]: Algorithm Design and Analysis; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*computations in finite fields, computations on matrices*

General Terms

Algorithms

Keywords

Linear system, rank profile, rank, sparse matrix

1. INTRODUCTION

Consider the following problems on an $n \times m$ input matrix A over a finite field K.

- LinSys: Given a $b \in \mathsf{K}^{n\times 1}$, compute a particular solution $x \in \mathsf{K}^{m\times 1}$ to $Ax = b$, or a certificate of inconsistency [6]: a row vector $u \in \mathsf{K}^{1\times n}$ such that $uA = 0$ and $ub \neq 0$.

- RankProfiles: Compute the rank r together with the lexicographically minimal lists $[i_1, i_2, \ldots, i_r]$ and $[j_1, j_2, \ldots, j_r]$ of row and column indices of A such that these rows and columns of A, respectively, are linearly independent.

The column rank profile of a matrix corresponds to the pivot locations in the reduced row echelon form, and is important information in applications such as Gröbner basis computations [5] and computational number theory [12].

We first discuss previous results and then discuss our new algorithms for solving the above problems. Running times of algorithms are given in terms of the number of required field operations from K.

Previous work

A lot of effort has been devoted to designing and analysing algorithms for problems LinSys and RankProfiles that have a running time that is sensitive to the rank r of A. Classical Gaussian elimination solves both problems in time $O(nmr)$. Using block recursive methods [3, 7] the running time is reduced to $O(nmr^{\omega-2})$ where ω is the exponent of matrix multiplication. A somewhat faster algorithm is possible when A is sparse [15]. Problem LinSys can be solved in time $O((n+m)r^2)$ using a so called oracle based variant [10, Section 2] of Gaussian elimination that examines a most $r+1$ rows and r columns of A. Note that if $r^2 < \min(m, n)$ then $(n+m)r^2 < nm$ so the running time of the oracle solver may be sublinear in the size of the input.

Now consider randomized algorithms. If only the rank of A is required then efficient preconditioners [1,8] give a Monte Carlo algorithm with running time $(r^\omega + nm)^{1+o(1)}$. Here, the $+o(1)$ in cost estimates captures some missing $\log n$ and $\log m$ factors. Heuristics and algorithms (with implementations) for computing the rank of very large (out of core) dense matrices modulo a prime p have been developed [9,11]. Now let $\mu(A)$ denote the time required to multiply a vector by A. For a sparse or structured matrix we may have $\mu(A) \in o(nm)$. In particular, if $|A|$ denotes the number of nonzero entries in A then we can take $\mu(A) \in O(|A|)$. Black-box approaches [8, 14] combined with early termination [4] can compute r in time $(\mu(A) \cdot r)^{1+o(1)}$; incorporating early termination into the black-box algorithm in [6] would seem to give a Las Vegas algorithm for LinSys with the same running time. In terms of the rank, none of the randomized approaches just mentioned recover the rank profile, or even determine a set of r linearly independent columns of A.

Our work in this paper is motivated by a recent breakthrough by Cheung, Kwok and Lau [2], who give an algorithm for computing the rank of A in time $(r^{\omega} + |A|)^{1+o(1)}$. Note that the $+|A|$ term in the cost estimate is optimal since changing a single entry of A might modify the rank. They also show how to compute a set of r linearly independent columns of A in the same time. However, the columns computed may not correspond to the column rank profile. For an extensive list of applications (with citations) of fast rank computation to combinatorial optimization and exact linear algebra problems we refer to [2].

New algorithms

In this paper we give new randomized algorithms for problems LinSys and RankProfiles that have improved running times in the special case when r is small compared to the row dimension n or column dimension m, or both. We give a Monte Carlo algorithm for RankProfiles that has running time bounded by $2r^3 + (r^2 + n + m + |A|)^{1+o(1)}$. We give a Las Vegas algorithm for LinSys that has running time $2r^3 + (r^2 + n + m + |R| + |C|)^{1+o(1)}$, where $|R|$ and $|C|$ are the number of nonzero entries in the subset of at most $r + 1$ rows and r columns of A, respectively, that are examined by the solver: at least $(m - r)(n - r - 1)$ entries of A are not examined. In later sections we develop the algorithms supporting these cost bounds directly for the general case of an input matrix $A \in \mathsf{K}^{n\times m}$ of (unknown) rank r. But to help explain our main technical contributions, we first consider the special case of an input matrix with full row rank, and then with full column rank.

Let $A \in \mathsf{K}^{n\times m}$ be an input matrix that has full row rank n, and consider the problem of computing the column rank profile of A. Gaussian elimination on A requires $O(n^2 m)$ time to compute a lower triangular matrix L such that, up to a row permutation, the matrix LA is upper triangular. The first few rows of such a transformation are shown in Figure 1. The columns of A corresponding to the pivot entries of LA (the first nonzero zero entry in each row) are thus linearly independent. But even if L is given, computing LA explicitly requires $O(n^2 m)$ time using standard matrix multiplication. Instead of computing LA explicitly, we show how to construct from the columns of A in time $|A|^{1+o(1)}$ a randomized data structure — a *linear independence oracle* — that allows to apply binary search to find the pivot locations in LA in a Monte Carlo fashion in time $(n^2 \log m)^{1+o(1)}$. This gives a Monte Carlo algorithm to find the column rank profile of a full row rank $A \in \mathsf{K}^{n\times m}$ in time $2n^3 + (n^2 + m + |A|)^{1+o(1)}$.

Now let $A \in \mathsf{K}^{n\times m}$ have full column rank m, and consider the problem of computing the row rank profile of A. Of course, we could simply apply the algorithm for column rank profile to the transpose of A. But with an eye to the general case (A with unknown rank) we develop an alternative method based on incorporating a linear independence oracle for the rows of A into the oracle linear solving algorithm [10]. We prove that the row rank profile of a full column rank $A \in \mathsf{K}^{n\times m}$ can be computed in a Monte Carlo fashion in time $2m^3 + (m^2 + n + |A|)^{1+o(1)}$ by running the linear solver algorithm with a right hand side vector b that is uniformly and randomly sampled from the column space of A.

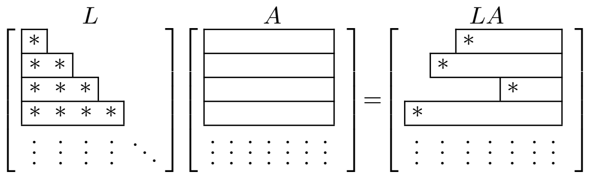

Figure 1: Example of pivot locations

The rest of the paper is organized as follows. In Section 2 we recall the deterministic oracle linear solving algorithm [10]. Section 3 shows that the deterministic oracle linear solver will recover the row and column rank profiles of A in a Monte Carlo fashion by choosing b to be a randomly and uniformly sampled vector from the column space of A. The linear independence oracle is described and analysed in Section 4. Sections 5 and 6 give faster algorithms for LinSys and RankProfiles for an input matrix $A \in \mathsf{K}^{n\times m}$ of (unknown) rank r by incorporating linear independence oracles for both the rows and columns of A. Section 7 concludes.

Throughout the paper we use the following notation. For a list $\mathcal{P}$ of distinct row indices and $\mathcal{Q}$ of distinct column indices, we write $A^{\mathcal{P}}$ to denote the submatrix of A consisting of rows $\mathcal{P}$, $A_{\mathcal{Q}}$ to denote the submatrix consisting of columns $\mathcal{Q}$, and $A^{\mathcal{P}}_{\mathcal{Q}}$ to denote the submatrix consisting of the intersection of rows $\mathcal{P}$ and columns $\mathcal{Q}$ of A.

2. ORACLE LINEAR SOLVING

Let $A \in \mathsf{K}^{n\times m}$ and $b \in \mathsf{K}^{n\times 1}$ be given. Oracle based elimination [10] is a variation of Gaussian elimination that either finds a particular solution to the linear system $Ax = b$ or proves that the system is inconsistent. The cost of the algorithm is $O((n+m)r^2)$ field operations from K, where r is the rank of A. This cost estimate, which can be $o(nm)$ for small values of r, can be achieved because the algorithm never examines more than r columns and $r + 1$ rows of the input matrix. The name was chosen because the target vector b is used as an oracle to determine a set of linearly independent rows of the matrix A.

The oracle based solver proceeds in stages for $s = 0, 1, \ldots$. The goal at stage s is to determine a list of row indices $\mathcal{P} = [i_1, i_2, \ldots, i_s]$ and column indices $\mathcal{Q} = [j_1, j_2, \ldots, j_s]$ such that

- $A^{\mathcal{P}} \in \mathsf{K}^{s\times m}$ has full row rank s,
- $A_{\mathcal{Q}} \in \mathsf{K}^{n\times s}$ has full column rank s, and
- $A^{\mathcal{P}}_{\mathcal{Q}} \in \mathsf{K}^{s\times s}$ is nonsingular.

To start the process for stage $s = 0$ the lists $\mathcal{P}$ and $\mathcal{Q}$ are simply initialized to be empty. To complete a stage s for some $s > 0$, the algorithm takes the following two steps.

1. If

$$b - A_{\mathcal{Q}} \cdot (A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \cdot b^{\mathcal{P}} \qquad (1)$$

is the zero vector then construct a particular solution to $Ax = b$ by setting x to be the zero vector except with entries at index $j_1, \ldots, j_{s-1}$ equal to entries at index $1, \ldots, s-1$ of $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \cdot b^{\mathcal{P}}$. Otherwise, let i_s be the index of the first nonzero entry of the column vector in (1) and go to step 2.

2. If
$$A^{[i_s]} - A^{[i_s]}_{\mathcal{Q}} \cdot (A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \cdot A^{\mathcal{P}} \tag{2}$$
is the zero vector then the system is inconsistent. A so called certificate of inconsistency [6] can be constructed by setting $u \in \mathsf{K}^{1\times n}$ to be the zero vector except with entry at index $i_1, \ldots, i_{s-1}$ equal to the entry at index $1, 2, \ldots, s-1$ of $-A^{[i_s]}_{\mathcal{Q}} \cdot (A^{\mathcal{P}}_{\mathcal{Q}})^{-1}$, and entry at index i_s to be 1. The vector u has the property that $uA = 0$ and $ub \neq 0$. Otherwise, set j_s to be the index of the first nonzero entry of the row vector in (2), update $\mathcal{P} = [i_1, i_2, \ldots, i_s]$ and $\mathcal{Q} = [j_1, j_2, \ldots, j_s]$, and proceed to stage $s+1$.

Note that $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1}$ need not be recomputed from scratch at each stage. Rather, the inverse for the next stage can be computed in $O(s^2)$ field operations from the inverse of the current stage by performing a rank 1 update. Since s is bounded by $r+1$, all the inverses required can be be computed in time $O(r^3)$. Similarly, computing the vectors in (1) and (2) costs $O(sn)$ and $O(sm)$ time, respectively, for an overall cost of $O((n+m)r^2)$.

Theorem 1. *There exists a deterministic algorithm* `OracleSolve`(A, b) *that takes as input an $A \in \mathsf{K}^{n\times m}$ and $b \in \mathsf{K}^{n\times 1}$, and returns as output either "consistent, x, $\mathcal{P}$, $\mathcal{Q}$" or "inconsistent, u, $\mathcal{P}$, $\mathcal{Q}$" as described above. The cost of the algorithm is $O((n+m)r^2)$ field operations from K.*

Although the overall cost estimate $O(r^3)$ for computing the required inverses does not dominate the running time of algorithm `OracleSolve`, it might dominate the running time for the improved variations of algorithm `OracleSolve` that we give in later sections. Thus, it will be useful here to derive the leading constant in this asymptotic bound for updating the inverses. Let $\mathcal{P} = [i_1, i_2, \ldots, i_{s-1}]$ and $\mathcal{Q} = [j_1, j_2, \ldots, j_{s-1}]$ be as at the start of stage s, and let i_s and j_s be as computed in steps 1 and 2. If we set $u = A^{\mathcal{P}}_{[j_s]} \in \mathsf{K}^{(s-1)\times 1}$, $v = A^{[i_s]}_{\mathcal{Q}} \in \mathsf{K}^{1\times(s-1)}$ and $d = A^{[i_s]}_{[j_s]} \in \mathsf{K}^{1\times 1}$, then at stage $s+1$ we need the inverse of
$$\left[\begin{array}{c|c} A^{\mathcal{P}}_{\mathcal{Q}} & u \\ \hline v & d \end{array}\right] \in \mathsf{K}^{s\times s}. \tag{3}$$

Lemma 2. *If the inverse $B := (A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \in \mathsf{K}^{(s-1)\times(s-1)}$ is precomputed, then the inverse of the matrix in (3), if it exists, is equal to*
$$\left[\begin{array}{c|c} B + (Bu)w(vB) & -(Bu)w \\ \hline -w(vB) & w \end{array}\right], \tag{4}$$
with $w = (d - v(Bu))^{-1}$, and can be computed in $6s^2 + O(s)$ field operations from K.

Proof. Correctness of the inverse is verified by direct computation. The total cost of the matrix×vector product Bu, the vector×matrix product vB, the outer product $(Bu)(vB)$, and the addition $B + (Bu)w(vB)$ is $6s^2 + O(s)$ field operations from K. The remaining dot products and scalar operations have cost $O(s)$. ☐

Since s is bounded by $r+1$, all the inverses required can be be computed in time $\sum_{s=1}^{r+1}(6s^2 + O(s)) = 2r^3 + O(r^2)$.

Remark 3. *The vector×matrix product $A^{[i_s]}_{\mathcal{Q}} \cdot (A^{\mathcal{P}}_{\mathcal{Q}})^{-1}$ in (2) for stage s is exactly the vector×matrix product vB used to update the inverse for stage $s+1$ in the proof of Lemma 2.*

Remark 4. *Assuming that Bu and vB in (4) have been precomputed, the matrix×vector product $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \cdot b^{\mathcal{P}}$ in (1) for stage $s+1$ can be computed in $O(s)$ field operations by applying the formula for $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1}$ shown in (4) to $b^{\mathcal{P}}$, using the fact that $Bb^{[i_1,\ldots,i_{s-1}]}$ is known from the previous stage.*

3. RANDOMIZED RANK PROFILES

In this section we establish that if b is uniformly and randomly sampled from the column space of A, the list of row indices $\mathcal{P}$ and the list of column indices $\mathcal{Q}$ returned by `OracleSolve`(A, b) will be the row and column rank profiles of A with high probability. (The list $\mathcal{Q}$ may need to be sorted in increasing order.) Algorithm 1 outlines this randomized algorithm for RankProfiles.

Algorithm 1 `RandomRankProfile`(A)
```
Require: $A \in \mathsf{K}^{n\times m}$
Ensure: $s \in \mathbb{Z}_{\geq 0}$, $\mathcal{P}$, $\mathcal{C}$
1: Choose a $w \in \mathsf{K}^{m\times 1}$ uniformly and randomly.
2: $b := Aw$
3: $*, *, \mathcal{P}, \mathcal{Q} :=$ OracleSolve$(A, b)$
4: $\mathcal{C} :=$ sort$(\mathcal{Q})$
5: Return length$(\mathcal{P})$, $\mathcal{P}$, $\mathcal{C}$
```

Lemma 5. *Let s, $\mathcal{P}$ and $\mathcal{C}$ be the output of Algorithm 1* `RandomRankProfile` *for an input matrix A of rank r. If $s = r$ then $\mathcal{C}$ is the column rank profile of A.*

Proof. If $s = r$ then the row space of $A^{\mathcal{P}}$ is equal to the row space of A. Now note that $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \cdot A^{\mathcal{P}}$ is equal to row permutation of the reduced row echelon form of A. Since the list $\mathcal{Q}$ corresponds to the column indices of the pivot entries in $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \cdot A^{\mathcal{P}}$ it is, up to sorting, equal to the column rank profile of A. ☐

Theorem 6. *The output $\mathcal{P}$ and $\mathcal{C}$ of Algorithm 1* `RandomRankProfile` *will be the row and column rank profile of A with probability at least $(1 - 1/\#\mathsf{K})^r$, where r is the rank of A.*

The following two lemmas contribute to the proof of Theorem 6.

Lemma 7. *Let $A \in \mathsf{K}^{n\times m}$ be a matrix of rank r and let $b \in \mathsf{K}^{n\times 1}$ be in the column space of A. Let*

- *$\bar{A} \in \mathsf{K}^{r\times m}$ be the submatrix of A comprised of the rank profile rows and let $\bar{\mathcal{P}}$ be the list of row indices returned by* `OracleSolve`$(\bar{A}, b)$*,*
- *$\mathcal{P}$ be the row indices returned by* `OracleSolve`(A, b)*.*

If $\bar{\mathcal{P}} = [1, 2, ..., r]$, then $\mathcal{P}$ is the row rank profile of A.

Proof. Assume $\bar{\mathcal{P}} = [1, 2, \ldots, r]$. Then the only difference in the computations done by the call `OracleSolve`(A, b)

compared to $\texttt{OracleSolve}(\bar{A}, b)$ occurs in step 1 when finding the first nonzero entry in the vector in (1): precisely the indices corresponding to rows not belonging to the rank profile will be skipped over since these rows of the augmented system $\left[\begin{array}{c|c} A & b \end{array}\right]$ are linearly dependent on the previous rows. $\square$

Lemma 8. *If $A \in \mathsf{K}^{n\times m}$ has full row rank n, and $w \in \mathsf{K}^{m\times 1}$ is chosen uniformly and randomly, then Aw is chosen uniformly and randomly from $\mathsf{K}^{n\times 1}$.*

Proof. Corresponding to $A \in \mathsf{K}^{n\times m}$ there always exists a nonsingular matrix $U \in \mathsf{K}^{m\times m}$ such that $AU = \left[\begin{array}{c|c} I_n & 0 \end{array}\right]$. Then $Aw = AU(U^{-1}w) = b$. When $w \in \mathsf{K}^{m\times 1}$ is uniformly and randomly chosen from K, $U^{-1}w$ is also uniformly and randomly chosen from $\mathsf{K}^{m\times 1}$. The result now follows by noting that b is equal to the first n entries of $U^{-1}w$. $\square$

Proof of Theorem 6. By Lemma 7, it will be sufficient to prove the theorem with input matrix $A \in \mathsf{K}^{r\times m}$ of full row rank r. When w is chosen uniformly and randomly from $\mathsf{K}^{m\times 1}$, vector $b = Aw \in \mathsf{K}^{r\times 1}$ is also chosen uniformly and randomly from $\mathsf{K}^{r\times 1}$ (Lemma 8). Since A is full rank, $\mathcal{P}$ is the row rank profile of A if $i_s = s$ for $1 \le s \le r$.

For some $s \ge 1$, suppose $\mathcal{P} = [i_1, i_2, ..., i_{s-1}]$ at the start of stage s has been computed correctly to be $[1, 2, \ldots, s-1]$. Then we claim that at stage s the probability that step 1 of the algorithm computes $i_s = s$ is equal to $1 - 1/\#\mathsf{K}$. To see this, note that the entry of (1) at index s is equal to

$$b[s] - A_{\mathcal{Q}}^{[s]} \cdot (A_{\mathcal{Q}}^{\mathcal{P}})^{-1} \cdot b^{\mathcal{P}}.$$

Since $b[s]$ is chosen uniformly and randomly from K, the probability that $b[s] = A_{\mathcal{Q}}^{[s]} \cdot (A_{\mathcal{Q}}^{\mathcal{P}})^{-1} \cdot b^{\mathcal{P}}$ is bounded by $1/\#\mathsf{K}$.

Since the entries in b are chosen independently, the success probability at each stage can be multiplied together to obtain $(1 - 1/\#\mathsf{K})^r$ for the probability that the algorithm returns $\mathcal{P} = [1, 2, \ldots, r]$.

Finally, if $\mathcal{P}$ is the rank profile of A, then by Lemma 5, $\mathcal{C}$ will be the column rank profile also. $\square$

For Algorithm 1, we remark that the output can satisfy $\text{length}(\mathcal{P}) = \text{rank}(A)$ even if $\mathcal{P}$ is not the rank profile of A. Consider the following example over $\mathsf{K} = \mathbb{Z}_3$:

$$A = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}, w = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b = Aw = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

At stage $s = 1$, the vector (1) in step 1 is equal to b. The first entry in (1) is zero, so $i_1 = 2$. Therefore, $\texttt{OracleSolve}(A, b)$ will return $\mathcal{P} = [2, 1]$ for the row rank profile, which is incorrect.

4. LINEAR INDEPENDENCE ORACLES

Throughout this section fix an $R \in \mathsf{K}^{r\times m}$ and let $v \in \mathsf{K}^{1\times r}$ be given. We describe the construction of a randomized binary tree data structure T that will allow us either determine if v is in the left nullspace of R, that is, if vR is the zero vector, or to find the first nonzero entry of $vR \in \mathsf{K}^{1\times m}$, in a Monte Carlo fashion in time $O(r \log m)$. We call T a *linear independence oracle* for R. Constructing a single linear independence oracle T for R costs $O(mr)$ field operations, which would seem to preclude its usefulness since we can compute $vR \in \mathsf{K}^{1\times m}$ explicitly in the same time. However, our algorithm for constructing T is online. We will show how to construct in overall $O(mr)$ time a sequence of linear independence oracles T_s for the submatrices $R^{[1,2,\ldots,s]}$ for R, $s = 1, 2, \ldots, r$.

Definition

By augmenting R with at most $m - 1$ zero columns we may assume without loss of generality that m is a power of two. A linear independence oracle T for R, shown in Figure 2, is a perfect binary tree of height $h := \log_2 m$, that is, with m leaf nodes. The nodes of the last level of the tree, from left to right, are associated with (i.e., store) the corresponding columns $R[1], R[2], \ldots, R[m]$ of R. In addition to R, the definition of the tree is based on a given list of elements $\alpha_1, \alpha_2, \ldots, \alpha_{m-1} \in \mathsf{K}$, which we associate to the internal nodes of T according to the level order traversal. In a bottom up fashion, for level $h-1, h-2, \ldots, 0$, associate to each internal node the vector obtained by adding the vector associated to the left child with α_* times the vector associated with the right child. For example, for the left child of the root we have $R_{1\sim\frac{m}{2}} = R_{1\sim\frac{m}{4}} + \alpha_2 R_{\frac{m}{4}+1\sim\frac{m}{2}}$. Note that by construction, $R_{a\sim b}$ is a vector that is a linear combination of $R[a], R[a+1], \ldots, R[b]$. We call T a *linear independence oracle* for R based on the parameters $\alpha_1, \ldots, \alpha_{m-1}$. In what follows it will be helpful to abuse notation somewhat and simply identify the nodes of T with the vectors stored at the node. For example, v is non-orthogonal with a node if the dot product of v with the node is nonzero.

Output

On the one hand, if v is in the left nullspace of R then v is orthogonal with the root $R_{1\sim m}$ as well since by construction $R_{1\sim m}$ is a linear combination of the columns of R. On the other hand, if $vR_{1\sim m}$ is nonzero, then because each internal node of T is a linear combination of the children, there must be a path from the root down to a leaf with v non-orthogonal to all nodes on the path.

Definition 9. *The* output of a linear independence oracle T with respect to a $v \in \mathsf{K}^{1\times r}$ *is either*

- *"v is in the left nullspace of R" if $vR_{1\sim m}$ is zero, or,*
- *the minimal j for which v is non-orthogonal with all nodes on the path from $R[j]$ back up to the root.*

Lemma 10. *The output of T with respect to v can be found in time $O(r \log_2 m)$.*

Proof. If $vR_{1\sim m}$ is zero we are done. Otherwise, use a straightforward recursive search starting at the root node. For example, if $vR_{1\sim\frac{m}{2}} \neq 0$ then recurse on the left subtree. If $vR_{1\sim m/2} = 0$ then necessarily $vR_{\frac{m}{2}+1\sim m} \neq 0$ so recurse on the right subtree. After arriving at a leaf node $R[j]$ report j. The cost is h dot products of length r. $\square$

Definition 11. *T is* correct with respect to *a $v \in \mathsf{K}^{1\times r}$ if either v is in the left nullspace of R, or the output of T with respect to v is the minimal j such that v is non-orthogonal to $R[j]$.*

Correctness of T will depend on the choice of the parameters $\alpha_1, \ldots, \alpha_{m-1}$ used during the construction. We first deal with construction of T before considering the probability of correctness.

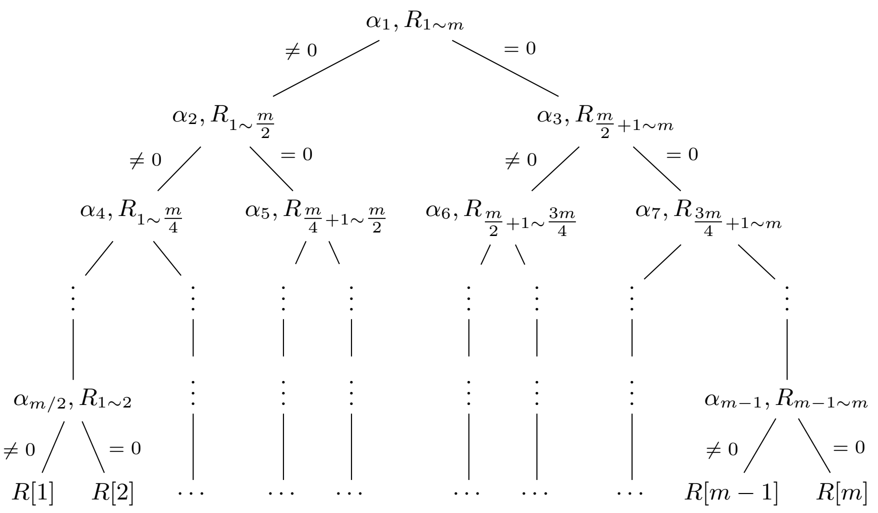
$\alpha_1, R_{1\sim m}$
$\neq 0$
$= 0$
$\alpha_2, R_{1\sim \frac{m}{2}}$
$\alpha_3, R_{\frac{m}{2}+1\sim m}$
$\alpha_4, R_{1\sim \frac{m}{4}}$
$\alpha_5, R_{\frac{m}{4}+1\sim \frac{m}{2}}$
$\alpha_6, R_{\frac{m}{2}+1\sim \frac{3m}{4}}$
$\alpha_7, R_{\frac{3m}{4}+1\sim m}$
$\alpha_{m/2}, R_{1\sim 2}$
$\alpha_{m-1}, R_{m-1\sim m}$
$R[1]$
$R[2]$
$R[m-1]$
$R[m]$

Figure 2: Oracle tree

Construction

THEOREM 12. *A sequence of linear independence oracles T_s for $R^{[1,2,\ldots,s]} \in \mathsf{K}^{s\times m}$, $s = 0, 1, \ldots, r$, all based on the parameters $\alpha_1, \ldots, \alpha_{m-1}$, can be constructed in total time $O(\min(mr, m + |R| \log_2 m))$, where $|R|$ denotes the number of nonzero entries of R.*

PROOF. We will construct T_s for $s = 0, 1, \ldots, r$ in succession. For efficiency, each column vector associated with a node in tree T_s should be represented as a singly linked list of nonzero elements, that is, each node in the list contains the index and the nonzero element of the vector stored at that index. Initialize T_0 to be a perfect binary tree with each node associated with a vector of length zero. The cost of this initialization is accounted for by the "$m+$" term in the cost estimate.

Given T_{s-1} for some $s > 0$, the linear oracle T_s is constructed from T_{s-1} and the next row vector $R^{[s]}$ in a bottom up fashion: augment the column vectors associated to the nodes in level i of T_{s-1} for $i = h, h-1, \ldots, 1$. Since there are $2m-1$ nodes in the tree the cost of obtaining T_s from T_{s-1} is bounded by $O(m)$. Since $1 \leq s \leq r$ the total cost is $O(mr)$.

But note that only the leaf nodes associated to nonzero elements of row $R^{[s]}$ may need to have their associated vectors modified. Similarly, internal nodes of the tree may need to be modified only if they had a child node that was modified. The cost of modifying each level is thus $|R^{[s]}|$. (See Example 13 below.) Since there are $1 + \log_2 m$ levels, the total cost of constructing T_s from T_{s-1} is $O(|R^{[s]}| \log m)$. Summing this estimate for $s = 1, 2, \ldots, r$, that is, over all rows of R, gives the claimed result. $\square$

EXAMPLE 13. *Suppose $r = 3$, $m = 8$ and*

$$R = \begin{array}{|cccccccc|} \multicolumn{1}{c}{1} & 2 & 3 & 4 & 5 & 6 & 7 & \multicolumn{1}{c}{8} \\ \hline & & * & & * & & & \\ \hline & & & * & * & & & \\ \hline & * & & & * & & & * \\ \hline \end{array} \in \mathsf{K}^{3\times 8},$$

with zero entries left blank and nonzero entries indicated with a $*$. All nodes in the initial tree T_0 store empty linked lists. The construction of T_1, T_2 and T_3 are shown in Figure 3. For brevity, for each node in the tree only the length of the list, corresponding to the number of nonzero entries in the vector, is indicated. The modified nodes (and their paths) are highlighted in dashed red.

Probability of correctness

LEMMA 14. *Let $\alpha_1, \ldots, \alpha_{m-1} \in \mathsf{K}$ be chosen uniformly and randomly. If T is a linear independence oracle for $R \in \mathsf{K}^{r\times m}$ based on $\alpha_1, \ldots, \alpha_{m-1}$, then T is correct with respect to v with probability at least $(1 - 1/\#\mathsf{K})^{\log_2 m}$.*

PROOF. If $vR = 0$ then T is correct independent of the choice of the $\alpha_1, \ldots, \alpha_{m-1}$, so assume $vR \neq 0$ and let j be the minimal index such that $vR[j] \neq 0$. Consider the path from $R[j]$ up to the root. If the sibling of $R[j]$ is $R[\bar{j}]$ then the parent of $R[j]$ is either $R[\bar{j}] + \alpha_* R[j]$ or $R[j] + \alpha_* R[\bar{j}]$, depending on whether or not j is even or odd, respectively. In either case, since $vR[j] \neq 0$ there is at most one choice of α_* such that v is orthogonal with the parent of $R[j]$. The same argument applies for the other internal nodes on the path from $R[j]$ up to the root. Since the α_* associated with each internal node are chosen uniformly and independently from K, and the number of internal nodes on the path is $h = \log_2 m$, the result follows. $\square$

Now, instead of a single $v \in \mathsf{K}^{1\times r}$, let a sequence of vectors $v_s \in \mathsf{K}^{1\times s}$ for $s = 1, 2, \ldots, r$ be given in addition to $R \in \mathsf{K}^{r\times m}$.

COROLLARY 15. *Let $\alpha_1, \ldots, \alpha_{m-1} \in \mathsf{K}$ be chosen uniformly and randomly. If $(T_s)_{1\leq s\leq r}$ is a sequence of linear independence oracles for $(R^{[1,2,\ldots,s]})_{1\leq s\leq r}$, all based on the same $\alpha_1, \ldots, \alpha_{m-1} \in \mathsf{K}$, then $(T_s)_{1\leq s\leq r}$ are correct with respect to $(v_s)_{1\leq s\leq r}$ simultaneously with probability at least $(1 - r/\#\mathsf{K})^{\log_2 m}$.*

PROOF. For $1 \leq s \leq r$, let $\bar{v}_s \in \mathsf{K}^{1\times r}$ be the vector obtained from v_s by augmenting with $r - s$ zeroes. Then T_s is correct with respect to v_s precisely when T is correct with respect to $\bar{v}_s$. For all $s \in [1, 2, \ldots, r]$ such that $\bar{v}_s R \neq 0$, consider the path from the leftmost leaf node in T that is

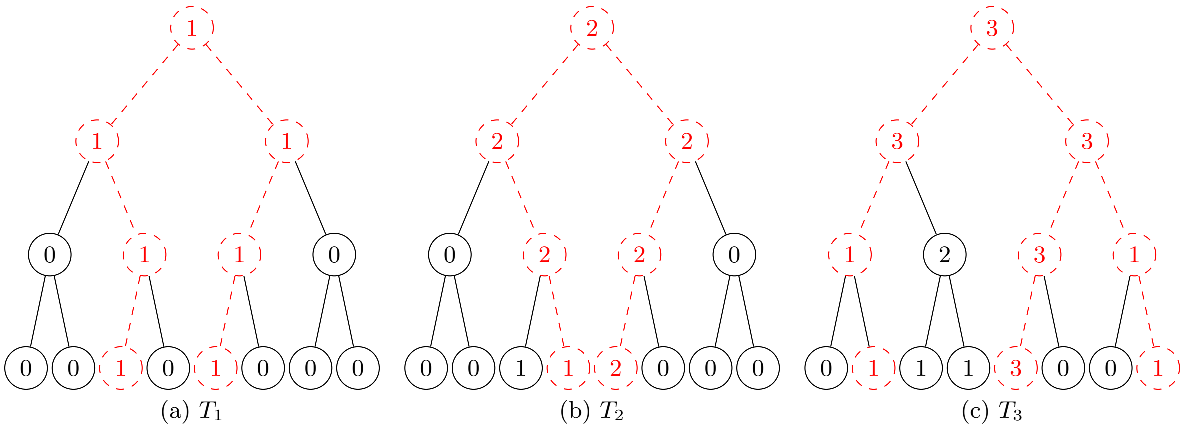

Figure 3: T_1, T_2, and T_3

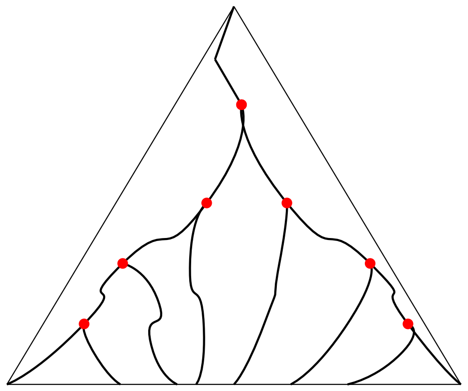

Figure 4: Non-orthogonal paths in a linear independence oracle

non-orthogonal to $\bar{v}$ back up to the root. There will be at most r such paths. (See Figure 4 for an example illustrating eight such paths.) Consider the choice of α_* for the nodes at level $i = h-1, h-2, \ldots, 0$ of T. In the best case, all the paths at level i from the leaf nodes back up to the root are independent, so we can multiply the probability of success at each internal node to get the estimate $(1 - 1/\#\mathsf{K})^r$ that all the α_* intersecting a path at level i are well chosen. However, as paths rise they may join implying that the same α_* has to be well chosen for more than one path. The worst case occurs near the root node when possibly all paths converge. In the worst case, we can bound from below the probability that a given α_* is good for all paths is at least $1 - r/\#\mathsf{K}$. Since there are $\log_2 m$ internal levels, and the α_* at each level are chosen independently, the result follows. $\square$

5. FASTER LINEAR SYSTEM SOLVING

The computations that dominated the running time in the deterministic oracle solver described in Section 2 were to determine, at stage $s = 1, 2, \ldots$, the first nonzero entry in $b - A_{\mathcal{Q}} \cdot (A_{\mathcal{Q}}^{\mathcal{P}})^{-1} \cdot b^{\mathcal{P}}$ and $A^{[i_s]} - A_{\mathcal{Q}}^{[i_s]} \cdot (A_{\mathcal{Q}}^{\mathcal{P}})^{-1} \cdot A^{\mathcal{P}}$ in steps 1 and 2, respectively. We can recast the computation in step 2 as follows. First compute

$$v_s := \left[\ -A_{\mathcal{Q}}^{[i_s]} \cdot (A_{\mathcal{Q}}^{\mathcal{P}})^{-1}\ \middle|\ 1\ \right] \in \mathsf{K}^{1\times s}$$

in $2s^2 + O(s)$ field operations. By Remark (3), the leading term $2s^2$ in this cost estimate is already counted in the worst case cost estimate for updating all the required inverses. Now set

$$R^{[1,2,\ldots,s]} := \left[\frac{A^{\mathcal{P}}}{A^{[i_s]}}\right] \in \mathsf{K}^{s\times m}.$$

We now need to assay if $v_s R^{[1,2,\ldots,s]} \in \mathsf{K}^{1\times m}$ is the zero vector, and find the index of the first nonzero element of $v_s R^{[1,2,\ldots,s]}$ otherwise: if we have precomputed a linear independence oracle T_s for $R^{[1,2,\ldots,s]}$ we can solve this problem in a Monte Carlo fashion in time $O(r \log m)$.

Since the situation in (1) is simply transposed the computation in step 1 can be recast similarly by computing

$$\bar{v}_s := \left[\frac{1}{-(A_{\mathcal{Q}}^{\mathcal{P}})^{-1} \cdot b^{\mathcal{P}}}\right]^T \in \mathsf{K}^{1\times s}$$

and setting

$$\bar{R}^{[1,2,\ldots,s]} = \left[\ b\ \middle|\ A_{\mathcal{Q}}\ \right]^T \in \mathsf{K}^{s\times n}.$$

By Remark 4, $\bar{v}_s$ can be computed in $O(s)$ field operations.

This gives the following algorithm.

1. Choose $\alpha_1, \ldots, \alpha_{m-1}, \beta_1, \ldots, \beta_{n-1}$ uniformly and randomly from K.

2. Use algorithm `OracleSolve` described in Section 2, but instead of explicitly computing the vectors in (1) for $s = 1, 2, \ldots$, initialize T_0 to be an oracle tree for the $0 \times m$ vector, and use the method supporting Theorem 12 to construct from T_{s-1} an $R^{[s]}$ and oracle tree T_s for $R^{[1,2,\ldots,s]}$ based on $\alpha_1, \ldots, \alpha_{m-1}$. Similarly, instead of explicitly computing the vectors in (2), use a sequence of oracle trees based on $\beta_1, \ldots, \beta_{n-1}$

3. Assay correctness of the output of the algorithm either by checking that $Ax = b$ or checking that $uA = 0$.

Corollary 15 bounds from below the probability that the linear independence oracles in step 2 will make exactly the same choices for $i_1, i_2, \ldots, i_s$ and $j_1, j_2, \ldots, j_s$ as the deterministic algorithm `OracleSolve` described in Section 2. We obtain the following result as a corollary of Lemma 10 (cost of using an oracle), Theorem 12 (constructing the sequences of oracles) and Corollary 15 (probability of success).

THEOREM 16. *There exists a randomized algorithm* `RandomOracleSolve`(A, b) *that takes as input an* $A \in \mathsf{K}^{n\times m}$ *and* $b \in \mathsf{K}^{n\times 1}$, *and returns as output either*

- *"FAIL, x, $\mathcal{P}$, $\mathcal{Q}$" with $Ax \neq b$, or*
- *"FAIL, u, $\mathcal{P}$, $\mathcal{Q}$" with $uA \neq 0$ and $ub \neq 0$, or*
- *"consistent, x, $\mathcal{P}$, $\mathcal{Q}$" with $Ax = b$, or*
- *"inconsistent, u, $\mathcal{P}$, $\mathcal{Q}$" with $uA = 0$ and $ub \neq 0$.*

Here, $\mathcal{P} = [i_1, \ldots, i_s]$ and $\mathcal{Q} = [j_1, \ldots, j_s]$ are such that $A^{\mathcal{P}}$ and $A_{\mathcal{Q}}$ have full row and column rank s, respectively. The algorithm has the following properties:

1. *$n + m - 2$ random choices from K are required.*
2. *If r is the rank of A, then with probability at least*
$$\left(1 - \frac{r}{\#\mathsf{K}}\right)^{\lceil \log_2 n\rceil + \lceil \log_2 m\rceil} \qquad (5)$$
FAIL is not returned and the output is identical to the output of algorithm `OracleSolver` *supporting Theorem 1.*
3. *The running time is bounded by*
$$2r^3 + O(r^2(\log n + \log m) + r(n+m))$$
and
$$2r^3 + O(r^2(\log n + \log m) + n + m + |A^{\mathcal{P}}|\log n + |A_{\mathcal{Q}}|\log m). \qquad (6)$$
field operations from K.

If $\#\mathsf{K}$ is too small we can work over a small field extension of $\#\mathsf{K}$ to ensure positive probability of success. The following corollary is obtained from (5) by substituting $r \leq \min(n, m)$ and using the fact that for any $x > 0$ and $y \in \mathbb{Z}_{>0}$ we have $1 - xy \leq (1-x)^y$.

COROLLARY 17. *If*
$$\#\mathsf{K} \geq 2\min(n, m)(\lceil \log_2 n\rceil + \lceil \log_2 m\rceil)$$
then the probability that algorithm `RandomOracleSolve` *does not return FAIL is at least* $1/2$.

By Corollary 17, the degree of the field extension required to ensure a probability of success at least $1/2$ is bounded by $O(\log n + \log m)$ in the worst case. If a field extension is required the cost of constructing and using the sequence of oracles increase by a multiplicative factor that is softly linear in $\log n + \log m$. However, the inverse computations via rank one updates during each phase of the oracle solver are still performed over the ground field and have overall cost bounded by $2r^3 + O(r^2)$ field operations. This gives the following result, valid over any finite field.

COROLLARY 18. *There exists a Las Vegas algorithm for problem* LINSYS *that has running time*
$$2r^3 + (r^2 + n + m + |R| + |C|)^{1+o(1)},$$
where R and C are the subsets of at most $r + 1$ rows and r columns of A that are examined by the algorithm.

6. FASTER RANK PROFILES

In the following theorem, let `ImprovedRankProfile` be identical to Algorithm 1 `RandomRankProfile` except with the call to the deterministic algorithm `OracleSolve` replaced with a call to the faster algorithm `RandomOracleSolve` supporting Theorem 16.

The following theorem follows as a corollary of Theorems 6 and 16.

THEOREM 19. *There exists a randomized algorithm* `ImprovedRankProfile`(A) *that takes as input an $A \in \mathsf{K}^{n\times m}$ and returns as output $s \in \mathbb{Z}_{\geq 0}$ together with lists $\mathcal{P} = [i_1, \ldots, i_s]$ and $\mathcal{C} = [j_1, \ldots, j_s]$ such that $A^{\mathcal{P}}$ and $A_{\mathcal{C}}$ have full row and column rank s, respectively. The algorithm has the following properties:*

1. *$n + 2m - 2$ random choices from K are required.*
2. *$\mathcal{P}$ and $\mathcal{C}$ will be the row and column rank profiles of A, respectively, with probability at least*
$$\left(1 - \frac{1}{\#\mathsf{K}}\right)^{r}\left(1 - \frac{r}{\#\mathsf{K}}\right)^{\lceil \log_2 n\rceil + \lceil \log_2 m\rceil}$$
where r is the rank of A.
3. *The running time is bounded by*
$$2r^3 + O(r^2(\log n + \log m) + nm)$$
and
$$2r^3 + O(r^2(\log n + \log m) + n + m + |A| + |A^{\mathcal{P}}|\log n + |A_{\mathcal{C}}|\log m) \qquad (7)$$
field operations from K.

The following two corollaries are very similar to Corollaries 17 and 18.

COROLLARY 20. *If*
$$\#\mathsf{K} \geq 2\min(n, m)(1 + \lceil \log_2 n\rceil + \lceil \log_2 m\rceil)$$
then the probability that `ImprovedRankProfile` *returns the correct result is at least* $1/2$.

COROLLARY 21. *There exists a Monte Carlo algorithm for problem* RANKPROFILES *that has running time bounded by* $2r^3 + (r^2 + n + m + |A|)^{1+o(1)}$.

7. CONCLUSIONS AND FUTURE WORK

For convenience, we have assumed that K is a finite field, which allowed us to choose elements uniformly and randomly from K, but it should not be difficult to extend algorithms `RandomOracleSolve` and `ImprovedLinSys` to work over any field, for example by choosing random elements from a sufficiently large subset of K.

Algorithm `ImprovedRankProfile` can compute the column rank profile of an $A \in \mathsf{K}^{n\times m}$ in a Monte Carlo fashion in time $2r^3 + (r^2 + |A|)^{1+o(1)}$, where $|A| \geq \max(n,m)$ is an upper bound on the number of nonzero entries in A. The $2r^3$ term in this cost estimate may seem unavoidable because the iterative nature of the algorithm and the needs to compute, for $s = 1, 2, 3, \ldots$, the vector $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \cdot b^{\mathcal{P}} \in \mathsf{K}^{s\times 1}$. Recently we have discovered an online algorithm for relaxed matrix inversion [13] that can be used to compute all these vectors in time $O(r^{\omega})$, provided that the final matrix $A^{\mathcal{P}} \in \mathsf{K}^{r\times m}$ has generic column rank profile. By incorporating the online matrix inversion algorithm together with a Toeplitz preconditioner [8] into algorithm `ImprovedRankProfile`, we can compute the row rank profile of a full column rank matrix A in time $(r^{\omega} + |A|)^{1+o(1)}$. Combined with the algorithm supporting [2, Theorem 2.11] for computing a maximal rank subset of of linearly independent columns, this gives an algorithm that computes the column rank profile of an arbitrary A with high probability in time $(r^{\omega} + |A|)^{1+o(1)}$. The full exposition of the relaxed online inversion algorithm together with its application to rank profile computation will be presented in a future paper.

8. REFERENCES

[1] L. Chen, W. Eberly, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Efficient matrix preconditioners for black box linear algebra. *Linear Algebra and its Applications*, 343–344:119–146, 2002. Special issue on *Infinite Systms of Linear Equations Finitely Specified*.

[2] H. Y. Cheung, T. C. Kwok, and L. C. Lau. Fast matrix rank algorithms and applications. *Journal of the ACM*, 60(5):733–751, 2013. Article No. 31.

[3] J.-G. Dumas, C. Pernet, and Z. Sultan. Simultaneous computation of the row and column rank profiles. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'13*, pages 181–188. ACM Press, New York, 2013.

[4] W. Eberly. Early termination over small fields. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'03*, pages 80–87. ACM Press, New York, 2003.

[5] J.-C. Faugère. A new efficient algorithm for computing Grobner basis (F4). *Journal of pure and applied algebra*, 139:61–88, 6 1999.

[6] M. Giesbrecht, A. Lobo, and B. D. Saunders. Certifying inconsistency of sparse linear systems. In O. Gloor, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'98*, pages 113—119. ACM Press, New York, 1998.

[7] C.-P. Jeannerod, C. Pernet, and A. Storjohann. Rank-profile revealing Gaussian elimination and the CUP matrix decomposition. *Journal of Symbolic Computation*, 56:56–58, 2013.

[8] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Proc. AAECC-9, Lecture Notes in Comput. Sci., vol. 539*, pages 29–38, 1991.

[9] J. P. May, B. D.Saunders, and Z. Wan. Efficient matrix rank computation with application to the study of strongly regular graphs. In J. P. May, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'07*, pages 277–284. ACM Press, New York, 2007.

[10] T. Mulders and A. Storjohann. Rational solutions of singular linear systems. In C. Traverso, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'00*, pages 242–249. ACM Press, New York, 2000.

[11] B. Saunders and B. Youse. Large matrix, small rank. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'09*, pages 317–324. ACM Press, New York, 2009.

[12] W. Stein. *Modular Forms, a Computational Approach. Graduate Studies in Mathematics.* American Mathematical Society, 2007.

[13] A. Storjohann and S. Yang. A relaxed algorithm for online matrix inversion, 2014. Poster available at `https://cs.uwaterloo.ca/~astorjoh/online.pdf`.

[14] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, IT-32:54–62, 1986.

[15] R. Yuster. Generating a d-dimensional linear subspace efficiently. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 467–470. Society for Industrial and Applied Mathematics, 2010.